

# Rapport du modèle de données orienté colonnes

## Equipe Brut Force(G7-S3)

*Les membres du groupe :*

- Ammour M'hena 1CS6
- Cherifi Rayane 1CS6
- Bouguerra Wail 1CS6
- Bouighebech Anouar 1CS6

22/01/2023

ESTIN

*Enseignant Encadrant : Mohamed Essaid KHANOUCHÉ*

## INTRODUCTION : Objectifs et contexte du projet

Ce rapport décrit le projet de mini-projet Planet Movies, qui a pour objectif de réaliser l'étude théorique et pratique d'un modèle de données NoSQL orienté colonnes. Ce modèle de données a été choisi pour sa flexibilité, sa scalabilité horizontale et sa capacité à gérer des données non structurées. Le projet a été réalisé par un groupe de 4 membres, CHERIFI Rayane , BOUGUERRA Wail ,BOUIGHEBECHE Anouar dirigé par le chef de projet M'hena AMMOUR,.

Dans le cadre de ce projet, nous avons mis en place une application web appelée "Planet Movies", similaire à IMDB, qui permet aux utilisateurs de consulter une liste de films, de filtrer les films en fonction de divers critères tels que la catégorie, l'année de production et les dernières sorties, et de consulter les détails d'un film en particulier, tels que le titre, l'année de production, la qualité, la description, l'affiche, la bande-annonce et les commentaires.

En outre, notre application présente des recommandations de films similaires à celui qui est affiché, ainsi qu'un système de commentaires où les utilisateurs peuvent discuter sur les films. Il y a aussi un système de favoris pour que les utilisateurs puissent sauvegarder leurs films préférés.

Pour la réalisation de cette application, nous avons utilisé les technologies suivantes:

- HTML, CSS et JavaScript pour le développement du front-end.

- Django, un framework Python, pour la gestion des routes, des vues et des formulaires de l'application.
- Django Cassandra Engine, une extension pour Django qui permet de gérer des bases de données Cassandra.
- Astra DB, un service cloud pour stocker notre base de données Cassandra.

En utilisant ces technologies, nous avons été en mesure de mettre en place un système de base de données NoSQL orienté colonnes qui est adapté aux besoins de notre application "Planet Movies". Ce rapport décrit les différentes étapes de notre projet, depuis l'étude théorique du modèle de données orienté colonnes jusqu'à l'implémentation pratique de l'application en passant par la modélisation des données, les requêtes utilisées pour la création, la mise à jour et la suppression des données, ainsi que les fonctionnalités développées pour l'interrogation, l'affichage et l'analyse des données.

## Bases de données NoSQL

NoSQL (Not Only SQL) est un terme utilisé pour décrire les systèmes de gestion de bases de données qui ne se basent pas sur la relationnelle, contrairement aux systèmes de gestion de bases de données relationnels (SQL). Les bases de données NoSQL sont souvent utilisées pour gérer des données volumineuses ou en évolution rapide, telles que les données de médias sociaux, de capteurs et de géolocalisation. Elles peuvent également être utilisées pour des besoins de scalabilité horizontale, de disponibilité élevée et de tolérance aux pannes. Il existe plusieurs types de bases de données NoSQL, dont les bases de données orientées colonnes, orientées clé-valeur, graphiques et documentaires.

## SQL vs NoSQL :

SQL (Structured Query Language) est un langage de programmation utilisé pour gérer les bases de données relationnelles. Les bases de données relationnelles utilisent des tables pour stocker les données, avec des relations définies entre les tables. Les requêtes SQL sont utilisées pour interagir avec ces tables et les données stockées à l'intérieur, telles

que la sélection, l'insertion, la mise à jour et la suppression de données. NoSQL, quant à lui, est un terme générique qui décrit les systèmes de gestion de bases de données qui ne suivent pas les normes des systèmes de gestion de bases de données relationnelles. Les bases de données NoSQL peuvent utiliser des modèles de données différents tels que clé-valeur, document, colonne, graphe, RDF, etc. Les requêtes NoSQL sont généralement moins structurées que les requêtes SQL et peuvent varier en fonction du modèle de données utilisé

Structure de données	Structurées (tables avec des colonnes et des lignes)	Non structurées (parfois de simples documents)
Schéma	Fortement typé, défini à l'avance	Souvent dynamique, peut être défini ou modifié à la volée
Requêtes	Langage de requête standard (SQL)	Langages de requête variés, souvent personnalisés pour chaque type de base de données
Consistance des données	ACID (Atomicité, Cohérence, Isolation, Durabilité)	BASE (Basically Available, Soft-State, Eventually Consistent)
Scalabilité	Verticale (ajout de ressources à une seule machine)	Horizontale (ajout de machines supplémentaires)
Performances	Meilleures pour les requêtes complexes, les transactions et les jointures	Meilleures pour les opérations de lecture/écriture de masse et les opérations de clé-valeur
Flexibilité	Moins flexible pour les données non structurées ou les schémas en évolution	Plus flexible pour les données non structurées ou les schémas en évolution

## Bases de données orientées colonnes

Les bases de données orientées colonnes sont conçues pour gérer des données volumineuses et en évolution rapide. Elles sont souvent utilisées pour les applications de

traitement de données massives, telles que les analyses de données de médias sociaux, les systèmes de surveillance de la santé et les systèmes de géolocalisation. Ces bases de données sont souvent plus performantes que les bases de données relationnelles pour les opérations de lecture et d'écriture de grandes quantités de données.

Les bases de données orientées colonnes sont également souvent utilisées pour des besoins de scalabilité horizontale, de disponibilité élevée et de tolérance aux pannes. Elles peuvent être facilement décomposées en plusieurs nœuds pour gérer des charges de traitement élevées.

Enfin, une base de données orientée colonnes est adaptée pour des cas d'utilisation nécessitant des requêtes de type "analytique" (agrégation, analyse de données) plutôt que des requêtes de type "transactionnel" (insertions, mise à jour).

## L'étude théorique du modèle de données orienté colonnes:

1. **La distribution de données sur plusieurs nœuds :** un concept clé dans les modèles de données orientés colonnes. Il consiste à répartir les données sur plusieurs serveurs ou nœuds, ce qui permet une scalabilité horizontale et une tolérance aux pannes. La scalabilité horizontale signifie que vous pouvez ajouter plus de nœuds pour gérer une croissance importante de la base de données, sans avoir à ajouter des ressources matérielles importantes. Cela permet également de gérer la disponibilité des données en cas de panne d'un nœud. La tolérance aux pannes signifie que le système peut continuer à fonctionner même si un ou plusieurs nœuds sont défaillants. Cela est possible grâce à la réplication des données sur plusieurs nœuds, ce qui permet de garantir que les données sont toujours accessibles, même en cas de défaillance d'un nœud. Pour la distribution de données sur plusieurs nœuds, il existe différentes approches de partitionnement de données, comme le partitionnement de données par clé, le partitionnement de données par rangée, le partitionnement de données par colonne, etc. En utilisant un système de gestion de bases de données orienté colonnes, la distribution de données sur plusieurs nœuds permet de gérer de grandes quantités de données de manière efficace, de garantir la disponibilité des données et de gérer la croissance à long terme de la base de données.

Voici un exemple pour illustrer le concept de distribution de données sur plusieurs nœuds : Imaginons qu'une entreprise de commerce électronique utilise un système de gestion de bases de données orienté colonnes pour stocker les informations de ses clients, les informations de commande et les informations de produits. Au départ, l'entreprise utilise un seul nœud pour stocker toutes ces informations. Avec le temps, le nombre de clients de l'entreprise augmente et il y a de plus en plus de commandes. La base de données devient de plus en plus grande et le nœud unique ne peut plus gérer la croissance de la base de données. Pour résoudre ce problème, l'entreprise décide de distribuer ses données sur plusieurs nœuds. Les informations de clients sont réparties sur un nœud, les informations de commandes sur un autre nœud et les informations de produits sur un troisième nœud. Cela permet à l'entreprise de gérer une croissance importante de la base de données sans avoir à ajouter des ressources matérielles importantes. De plus, en utilisant la réplication des données, l'entreprise peut garantir la disponibilité des données même en cas de panne d'un nœud. Ainsi, même si un nœud est défaillant, les données restent accessibles via les réplicas. Ainsi, en utilisant un système de gestion de bases de données orienté colonnes et en distribuant les données sur plusieurs nœuds, l'entreprise est en mesure de gérer efficacement les données et de garantir la disponibilité des données même en cas de pannes.

2. **Les réplicas** sont une autre caractéristique importante des modèles de données orientés colonnes. Les réplicas sont des copies des données qui sont stockées sur plusieurs nœuds. Cela permet de garantir la disponibilité des données en cas de panne d'un nœud. Il existe différents types de réplication, chacun ayant des avantages et des inconvénients :

**La réplication simple** : Il s'agit d'une réplication de données sur un seul serveur de réplication. Cette approche est simple à mettre en place, mais elle n'offre pas une haute disponibilité en cas de panne d'un nœud.

**La réplication à plusieurs maîtres** : Il s'agit d'une réplication de données sur plusieurs serveurs de réplication, chacun étant capable de gérer les mises à jour. Cette approche offre une haute disponibilité, mais elle nécessite une gestion des conflits lorsque les mises à jour sont effectuées sur plusieurs nœuds en même temps.

**La réplication à plusieurs niveaux** : Il s'agit d'une réplication de données sur plusieurs niveaux, chacun étant destiné à une utilisation spécifique (par exemple,

une réPLICATION pour les sauvegardes, une réPLICATION pour les analyses, une réPLICATION pour les requêtes, etc.). Cette approche offre une haute disponibilité et une flexibilité, mais elle nécessite une gestion complexe des réPLICAS. En utilisant la réPLICATION des données, les modèles de données orientés colonnes peuvent garantir la disponibilité des données même en cas de panne d'un nœud. Cela permet également de gérer les pics de charge en répartissant les requêtes sur plusieurs nœuds.

**Voici un exemple pour illustrer le concept de réPLICATION :** Imaginons qu'une entreprise utilise un système de gestion de bases de données orienté colonnes pour stocker les informations de ses employés. Les informations sont stockées sur un seul nœud, mais l'entreprise souhaite garantir la disponibilité des données en cas de panne d'un nœud. Pour résoudre ce problème, l'entreprise décide de mettre en place une réPLICATION à plusieurs maîtres. Les informations sont répliquées sur trois nœuds différents, chacun étant capable de gérer les mises à jour. Si un nœud est défaillant, les informations restent accessibles via les deux autres nœuds de réPLICATION. De plus, si l'entreprise souhaite effectuer une mise à jour sur les informations, elle peut le faire sur l'un des nœuds de réPLICATION, ce qui garantit que les informations sont toujours à jour. Ainsi, en utilisant la réPLICATION à plusieurs maîtres, l'entreprise est en mesure de garantir la disponibilité des données en cas de panne d'un nœud, tout en garantissant que les informations sont toujours à jour.

3. **Les index de colonnes** sont un autre concept clé des modèles de données orientés colonnes. Les index de colonnes sont des structures de données qui permettent d'optimiser les requêtes en facilitant la recherche de données spécifiques dans les tables de la base de données. Les index de colonnes sont similaires aux index que l'on peut trouver dans les systèmes de gestion de bases de données relationnelles, mais ils sont conçus pour fonctionner avec les modèles de données orientés colonnes. Les index de colonnes permettent de stocker des valeurs de colonnes spécifiques dans une structure de données triée, ce qui permet de rechercher rapidement des valeurs spécifiques dans les tables. Cela permet d'optimiser les performances des requêtes en réduisant le nombre de lignes qui doivent être parcourues pour trouver les données recherchées. Il existe plusieurs types d'index de colonnes, chacun ayant des avantages et des inconvénients :

**index de colonnes non uniques** : ces index permettent de stocker les valeurs de colonnes en double

**index de colonnes uniques** : ces index n'autorisent pas les valeurs de colonnes en double

Il est important de noter que l'utilisation des index de colonnes peut améliorer les performances des requêtes, mais cela peut également augmenter la complexité de la gestion de la base de données et augmenter la consommation de ressources. Il est donc important de bien évaluer les besoins de votre application avant de choisir d'utiliser des index de colonnes

**Voici un exemple pour illustrer le concept des index de colonnes** en utilisant notre site de movie review "Planet Movie" : Planet Movie stocke les informations sur les films et les critiques dans une table appelée "films". Cette table contient des colonnes telles que "Title", "Year\_of\_prod" . Souvent, les utilisateurs souhaitent rechercher des films en fonction de leur nom ou de leur année de sortie. Pour optimiser les performances des requêtes, Planet Movie décide d'utiliser des index de colonnes sur les colonnes "Title" et "Year\_of\_prod". Lorsqu'un utilisateur effectue une recherche pour un film spécifique, le système de gestion de bases de données utilise l'index de colonne sur la colonne "Title" pour rapidement trouver les lignes correspondantes dans la table "films", ce qui permet de réduire le nombre de lignes qui doivent être parcourues pour trouver les données recherchées. De la même manière, si un utilisateur effectue une recherche pour des films sortis pendant une année spécifique, le système de gestion de bases de données utilise l'index de colonne sur la colonne "Year\_of\_prod" pour rapidement trouver les lignes correspondantes dans la table "films", ce qui permet de réduire le nombre de lignes qui doivent être parcourues pour trouver les données recherchées.

Ainsi , en utilisant des index de colonnes sur les colonnes "Title" et "Year\_of\_prod", Planet Movie est en mesure d'optimiser les performances des requêtes et de fournir des résultats de recherche rapides pour ses utilisateurs. Cela améliore l'expérience utilisateur et permet de gérer efficacement les données de la base de données en termes de recherche. Il est important de noter que cela nécessite une gestion supplémentaire pour maintenir les index de colonnes à jour lorsque des modifications sont apportées aux données, mais cela peut s'avérer être un compromis acceptable pour améliorer les performances des requêtes.

4. **La structure de données** est l'organisation logique des données dans un système de gestion de bases de données. Elle décrit comment les données sont stockées,

comment elles sont reliées entre elles et comment elles sont accédées. Les modèles de données NoSQL, tels que les modèles orientés colonnes, utilisent des structures de données différentes de celles des systèmes de gestion de bases de données relationnelles. Dans les modèles de données orientés colonnes, les données sont stockées dans des tables qui ont des colonnes, et non pas des champs comme dans les modèles relationnels, chaque ligne de ces tables contient des valeurs pour chaque colonne. Ces colonnes peuvent être organisées en familles de colonnes, qui regroupent des colonnes similaires.

Prenons l'exemple de Planet Movie : Il utilise une structure de données orientée colonnes pour stocker les informations sur les films et les critiques. Il a une table appelée "Movie" qui contient des colonnes telles que "Title", "Year\_of\_prod" . Les données sont stockées sous forme de lignes dans cette table avec une valeur pour chaque colonne. Il a également une famille de colonnes appelée "acteurs" qui regroupe des colonnes telles que "Acteur\_name" et "Acteur\_Role". La structure de données orientée colonnes utilisée par Planet Movie lui permet de stocker et de gérer efficacement les données sur les films et les critiques. Elle lui permet également de faciliter la mise à jour et la recherche de données spécifiques en utilisant des index de colonnes et des requêtes ciblant les familles de colonnes appropriées.

5. **La gestion des métadonnées** : Dans les modèles de données orientées colonnes, les métadonnées peuvent être stockées dans des colonnes dédiées dans la table principale ou dans des tables séparées. Les colonnes dédiées aux métadonnées peuvent inclure des informations telles que la date de création, la date de modification, l'utilisateur qui a créé ou modifié les données, etc.

Par exemple, dans le cas de Planet Movie, la table principale pour les films pourrait avoir des colonnes dédiées aux métadonnées telles que "date\_creation" et "utilisateur\_creation" pour stocker la date et l'utilisateur qui ont créé l'enregistrement de film. Il pourrait également avoir des colonnes "date\_modification" et "utilisateur\_modification" pour stocker la date et l'utilisateur qui ont modifié l'enregistrement de film. Il est important de noter que ces colonnes dédiées aux métadonnées sont généralement automatiquement mises à jour lorsque des modifications sont apportées à l'enregistrement de film, ce qui permet de suivre l'historique des modifications apportées aux films. Il est également possible d'utiliser une table séparée pour stocker les métadonnées, cette table pourrait avoir une colonne "id\_film" pour lier les métadonnées à un

film spécifique, et des colonnes pour stocker les informations de métadonnées telles que la date de création, la date de modification, l'utilisateur qui a créé ou modifié les données, etc. En utilisant des colonnes dédiées aux métadonnées ou une table séparée, il est possible de facilement suivre et gérer les informations de métadonnées pour mieux comprendre les données stockées dans la base de données orientée colonne.

## NOSQL OU SQL qui est le plus approprié :

**Sujet 1 :** Le sujet particulier est la gestion des données de localisation GPS en temps réel pour les applications de surveillance de la flotte de véhicules:

Les données de localisation GPS incluent des informations telles que la latitude, la longitude, la vitesse et l'heure de chaque véhicule. Ces données doivent être collectées en temps réel et mises à disposition immédiatement pour une utilisation par les applications de surveillance.

L'utilisation d'un système de gestion de bases de données relationnel (SQL) pour stocker et gérer ces données de localisation GPS serait peu adaptée ou inappropriée pour plusieurs raisons: Les données de localisation GPS sont générées en grande quantité et à une fréquence élevée, ce qui pourrait causer des problèmes de performance pour les requêtes SQL. Les données de localisation GPS sont souvent liées à des données de temps en temps réel, ce qui pourrait causer des problèmes de synchronisation pour les systèmes de gestion de bases de données relationnelles. Les données de localisation GPS sont souvent utilisées pour générer des cartes en temps réel et des visualisations de données, ce qui pourrait causer des problèmes pour les systèmes de gestion de bases de données relationnelles qui sont pas conçus pour gérer des données spatiales. D'un autre coté, l'utilisation d'un système de gestion de bases de données NoSQL tel que les bases de données orientées colonnes ou orientées document serait plus adaptée pour stocker et gérer ces données de localisation GPS en temps réel. Ces systèmes de gestion de bases de données NoSQL sont conçus pour gérer des données de grande quantité et de grande vitesse, et peuvent facilement gérer des données de temps en temps réel

**sujet 2 :** Voici un exemple de comment l'utilisation d'un SGBD NoSQL serait plus adaptée pour gérer les données de notre site "Planet Movie" :

- Le site "Planet Movie" permet aux utilisateurs de noter et de commenter les films

qu'ils ont regardés. Ces données sont générées en grande quantité et à une fréquence élevée, ce qui pourrait causer des problèmes de performance pour les requêtes SQL. Les données de notation et de commentaires sont souvent liées à des données de temps en temps réel, comme la date et l'heure où l'utilisateur a noté ou commenté le film. Ce qui pourrait causer des problèmes de synchronisation pour les systèmes de gestion de bases de données relationnelles.

- Le site "Planet Movie" a besoin de générer des statistiques et des analyses sur les données de notation et de commentaires, comme les films les plus populaires et les utilisateurs les plus actifs. Ce qui pourrait causer des problèmes pour les systèmes de gestion de bases de données relationnelles qui ne sont pas conçus pour gérer des données statistiques. En utilisant un système de gestion de bases de données NoSQL tel que les bases de données orientées colonnes ou orientées document.
- le site "Planet Movie" pourrait facilement gérer les données de notation et de commentaires en temps réel, les stocker de manière efficace et générer des statistiques et des analyses en temps réel pour améliorer l'expérience utilisateur.
- Le site "Planet Movie" permet aux utilisateurs de rechercher des films par catégories, telles que comédie, action, aventure, etc. Ces données sont liées à des données de films et de catégories, ce qui pourrait causer des problèmes de performance pour les requêtes SQL dans les systèmes de gestion de bases de données relationnelles. Les utilisateurs peuvent rechercher des films par plusieurs catégories en même temps, ce qui pourrait causer des problèmes de performance pour les systèmes de gestion de bases de données relationnelles qui ne sont pas conçus pour gérer des requêtes de recherche multi-critères.
- Le site "Planet Movie" a besoin de générer des statistiques sur les catégories de films les plus populaires et les films les plus populaires par catégories, ce qui pourrait causer des problèmes pour les systèmes de gestion de bases de données relationnelles qui ne sont pas conçus pour gérer des données statistiques. En utilisant un système de gestion de bases de données NoSQL tel que les bases de données orientées colonnes,
- le site "Planet Movie" pourrait facilement gérer les données de recherche de films par catégories en utilisant des index de colonnes pour la recherche rapide et efficace, et générer des statistiques sur les catégories de films les plus populaires et les films les plus populaires par catégories pour améliorer l'expérience utilisateur.

## **Apache Cassandra:**

Apache Cassandra est un système de gestion de base de données NoSQL distribué open-source qui fonctionne sur le principe de la base de données orientée colonne. Il est conçu pour gérer des grandes quantités de données réparties sur plusieurs nœuds sans point unique de défaillance. Il est également connu pour sa capacité à gérer des requêtes à haut débit et à haute disponibilité. Cassandra est basé sur un modèle de données de colonnes dynamiques qui permet de stocker des données de différents types, comme des textes, des nombres, des dates et des booléens, dans des colonnes indépendantes. Il prend en charge des opérations de lecture et d'écriture à haute performance, ainsi que des index de colonnes pour améliorer les performances de requête. Il fournit également un système de réPLICATION de données pour assurer la haute disponibilité des données, ainsi qu'un système de gestion des métadonnées pour faciliter la gestion des données. Il est utilisé par de nombreuses grandes entreprises pour gérer des applications à grande échelle, telles que les réseaux sociaux, les sites de commerce électronique et les systèmes de recommandation

## **Cassandra query language(cql) :**

CQL (Cassandra Query Language) est le langage de requête utilisé pour interagir avec les bases de données Cassandra. Il est similaire à SQL (Structured Query Language) utilisé pour les bases de données relationnelles, mais il a quelques différences importantes. CQL est utilisé pour créer, lire, mettre à jour et supprimer des données dans une base de données Cassandra. Il prend en charge la plupart des opérations SQL standard, telles que la sélection, l'insertion, la mise à jour et la suppression de données. CQL est également utilisé pour créer et gérer des tables, des index, des types de données et des utilisateurs dans une base de données Cassandra. Il est généralement utilisé avec un client CQL, comme cqlsh, pour exécuter des commandes CQL contre une base de données Cassandra..

Voici quelques exemples de requêtes en CQL (Cassandra Query Language) :

1. Créer une table :

```
CREATE TABLE users (
    user_id int PRIMARY KEY,
```

```
username text,  
email text  
);
```

2. Insérer des données dans une table :

```
INSERT INTO users (user_id, username, email) VALUES (1, 'johndoe',  
'johndoe@example.com');
```

3. Sélectionner des données d'une table :

```
SELECT * FROM users WHERE user_id = 1;
```

4. Mettre à jour des données d'une table :

```
UPDATE users SET email = 'johndoe123@example.com' WHERE user_id = 1;
```

5. Supprimer des données d'une table :

```
DELETE FROM users WHERE user_id = 1;
```

6. Créer un index sur une colonne :

```
CREATE INDEX ON users (username);
```

7. Sélectionner des données avec un index :

```
SELECT * FROM users WHERE username = 'johndoe';
```

8. Supprimer une table :

```
DROP TABLE users;
```

## La préparation de l'environnement concernant le SGBD NoSQL:

Voici les étapes détaillées pour préparer l'environnement concernant l'utilisation de Django Cassandra Engine (SGBD NoSQL) :

1. **Installation de Cassandra:** Pour installer Cassandra sur votre système, vous pouvez suivre les instructions officielles disponibles sur le site Web de Cassandra. Il existe plusieurs options pour installer Cassandra, comme l'utilisation d'un package manager (apt ou yum), ou l'utilisation d'un installer binaire. Vous devez vous assurer que Cassandra est correctement installé en vérifiant la version en utilisant la commande `cassandra -v`.
2. **Configuration de Cassandra:** Pour configurer Cassandra, vous devez éditer les fichiers de configuration appropriés, tels que `cassandra.yaml` et `cassandra-env.sh`. Ces fichiers se trouvent généralement dans le répertoire d'installation de Cassandra (par exemple, `/etc/cassandra/`). Vous devez configurer les paramètres tels que les adresses IP des noeuds, les ports, les options de journalisation, les options de mémoire, etc.
3. **Installation de Django Cassandra Engine:**

Pour installer Django Cassandra Engine, vous devez utiliser pip ou un autre gestionnaire de paquets en utilisant la commande `pip install django-cassandra-engine`.

Vous devez vous assurer que Django Cassandra Engine est correctement installé en vérifiant la version en utilisant la commande : `pip show django-cassandra-engine`.

4. **Configuration de Django Cassandra Engine :**

Pour configurer Django Cassandra Engine, vous devez ajouter les paramètres appropriés dans le fichier `settings.py` de Django. Ces paramètres incluent les options de connexion à la base de données Cassandra, les options de

modèles, les options de migration, etc. Vous devez également ajouter 'cassandra\_engine' dans l'INSTALLED\_APPS de votre fichier settings.py

```
INSTALLED_APPS = ['django_cassandra_engine'] + INSTALLED_APPS
```

```
SESSION_ENGINE = 'django_cassandra_engine.sessions.backends.db'
```

## 5. Mise en place de la base de données :

Pour mettre en place les tables et les colonnes appropriées pour votre application, vous devez utiliser les commandes **Cassandra (CQL)** pour créer les **keyspaces et les tables**, ainsi que les modèles Django Cassandra Engine pour décrire les colonnes. Il est important de noter que les modèles Cassandra ne sont pas les mêmes que les modèles traditionnels de Django et ils doivent être conçus en conséquence , voila un exemple de creation de modeles sur django en utilisant cassandra:

```
 29 class Movie(DjangoCassandraModel,models.Model):
30     class Meta:
31         get_pk_field='idfilm'
32
33     idfilm=columns.UUID(primary_key=True , default = uuid.uuid4)
34     title=columns.Text(max_length=200,required=True)
35     image=columns.Text(max_length=1000)
36     movie_length=columns.Integer(50)
37     category=columns.Text(primary_key=True,required=True,partition_key=True)
38     status=columns.Text(required=True)
39     language=columns.Text(required=True)
40     year_of_prod=columns.DateTime()
41     added_at=columns.DateTime(default=datetime.now)
42     views_nb=columns.Integer(50)
43
44
45     def __str__(self):
46         return self.title
47
48
```

Vous pouvez utiliser les commandes CQL pour vérifier si les tables et les colonnes ont été créées correctement . L'équivalent de cette commande en cql(cassandra query language) :

```
CREATE TABLE movie (
    idfilm uuid PRIMARY KEY,
    title text,
```

```

image text,
movie_length int,
category text, status text,
language text,
year_of_prod timestamp,
added_at timestamp,
views_nb int )WITH partition by (category);

```

- Connexion à Astra DB: Pour stocker les données de votre application dans Astra DB, vous devez vous connecter à Astra DB en utilisant les informations de connexion appropriées, telles que l'adresse IP, le nom d'utilisateur et le mot de passe. Vous pouvez utiliser un client Cassandra pour vous connecter à Astra DB, tel que cqlsh. Vous pouvez également utiliser l'interface web d'Astra DB pour vérifier si les données ont été stockées correctement.

The screenshot shows two side-by-side web pages. On the left is the 'Sign In' page for DataStax Astra DB, featuring social login options for GitHub and Google, and fields for Work Email and Password. On the right is the main Astra DB landing page, which highlights it as a 'Multi-cloud DBaaS Built on Apache Cassandra™'. It lists three key benefits: starting in minutes, building faster with various APIs, and deploying on multiple clouds. A callout box introduces 'Astra Streaming!', describing it as an open event-streaming service powered by Apache Pulsar™. A purple speech bubble icon is visible in the bottom right corner of the landing page.

- Astra db nous donne la possibilité d'utiliser la console cql pour executer des requetes cql :

The screenshot shows the DataStax Astra web interface. At the top, a blue banner says "NEW We've been working on a new Astra Experience — try it out today! You can always turn off the new experience if you prefer this one." with a "Enable New UI" button. The header includes the DataStax Astra logo, a "Live Chat" button, and a user profile for "rayane CHERIFI".

The main area has a left sidebar with "Current Organization r\_cherifi@estin.dz". It contains sections for "Dashboard", "Databases" (with "workshops", "techblog", and "planet" listed), "Streaming", and "Sample App Gallery". There are also links for "Documentation" and "Help Center", and a "Get Credits" section.

The right side is the "CQL Console" tab, which is active. It shows a CQL session connected to "cassandra.ingress:9042". The session output includes:

```
Connected as r_cherifi@estin.dz.
Connected to cndb at cassandra.ingress:9042.
[cqlsh 6.8.0 | Cassandra 4.0.0.6816 | CQL spec 3.4.5 | Native protocol v4]
Use HELP for help.
token@cqlsh> use planet;
token@cqlsh:planet> desc tables;

comment fav_film movie4 post_model session user

token@cqlsh:planet>
```

- Ici on peut très bien prévisualisé le nombre de lecture et d'écriture effectuer sur la base de donnée

**Astra**

We've been working on a new Astra Experience — try it out today! You can always turn off the new experience if you prefer this one.

Current Organization: r\_cherifi@estin.dz

- Dashboard
- Databases [Create Database](#)
  - workshops
  - techblog
  - planet
- Streaming [Create Stream](#)

**Sample App Gallery**  
Looking to get up and running? Get started with a sample app!

Other Resources
 

- Documentation
- Help Center

Invite Friends, Get Credits  
You and your friend both get \$25!

Dashboard / Serverless Databases **planet** [Active]

Overview Health Connect CQL Console CDC Settings

General / DSE Cluster Condensed

Last 30 minutes [Load Data](#) [Connect](#)

**Request Overview**

Requests Combined

Request Errors

Read Failures, Read Timeouts, ReadUnavailable, CAS Read Failures, CAS Read Timeouts, Write Failures, Write Timeouts, WriteUnavailable, CAS Write Failures, CAS Write Timeouts, CAS WriteUnavailable, Range Failures, Range Timeouts, RangeUnavailable

**Writes**

Write Latency

Write Size Distribution

80 B, 60 B, 40 B

We've been working on a new Astra Experience — try it out today! You can always turn off the new experience if you prefer this one.

Current Organization: r\_cherifi@estin.dz

- Dashboard
- Databases [Create Database](#)
  - workshops
  - techblog
  - planet
- Streaming [Create Stream](#)

**Sample App Gallery**  
Looking to get up and running? Get started with a sample app!

Other Resources
 

- Documentation
- Help Center

Invite Friends, Get Credits  
You and your friend both get \$25!

Dashboard / Serverless Databases **planet** [Active]

Overview Health Connect CQL Console CDC Settings

Usage For Current Billing Period for planet

Status Active

Read Requests	Write Requests	Storage Consumed	Data Transfer
2.1k	121	543.51 KB	142.56 MB

Regions

Our Pay as you go plan allows you to add multiple regions.

Unlock Multi-Region

Provider	Area	Region	Region Name	Datacenter ID	Region Availability
Google Cloud	North America	us-east1	Moncks Corner, South Carolina	b8da713d-32e2-419e-bf8f-95ad70898f80-1	Online

Regions

Our Pay as you go plan allows you to add multiple regions.

Unlock Multi-Region

Provider	Area	Region	Region Name	Datacenter ID	Region Availability
Google Cloud	North America	us-east1	Moncks Corner, South Carolina	b8da713d-32e2-419e-bf8f-95ad70898f80-1	Online

Keystpaces

Managing multiple applications? Consider keeping each application in a separate keyspace – whatever a [keyspace](#) is.

Add Keyspace

Keyspace
planet

## Les requêtes CQL pour créer ces tables de notre site web:

Voici les requêtes de création de tables en utilisant le langage CQL:

1. Pour la table "PostModel":

```
CREATE TABLE postmodel (
    id uuid PRIMARY KEY,
    title text,
    body text,
    created_at timestamp
);
```

2. Pour la table "Movie4":

```
CREATE TABLE movie4 (
    idfilm uuid PRIMARY KEY,
    title text,
    image text,
    category text,
    status text,
    language text,
    year_of_prod int,
    added_at timestamp,
    views_nb int,
    duration text,
```

```
        description text,  
        video_id text,  
partition_key (category, idfilm)  
  
);WITH CLUSTERING ORDER BY (idfilm DESC);
```

3. Pour la table "user":

```
CREATE TABLE user (  
        id uuid PRIMARY KEY,  
        username text,  
        password text,  
        email text  
);
```

4. Pour la table "Comment":

```
CREATE TABLE comment (  
        id uuid PRIMARY KEY,  
        text text,  
        user_id text,  
        film_id uuid,
```

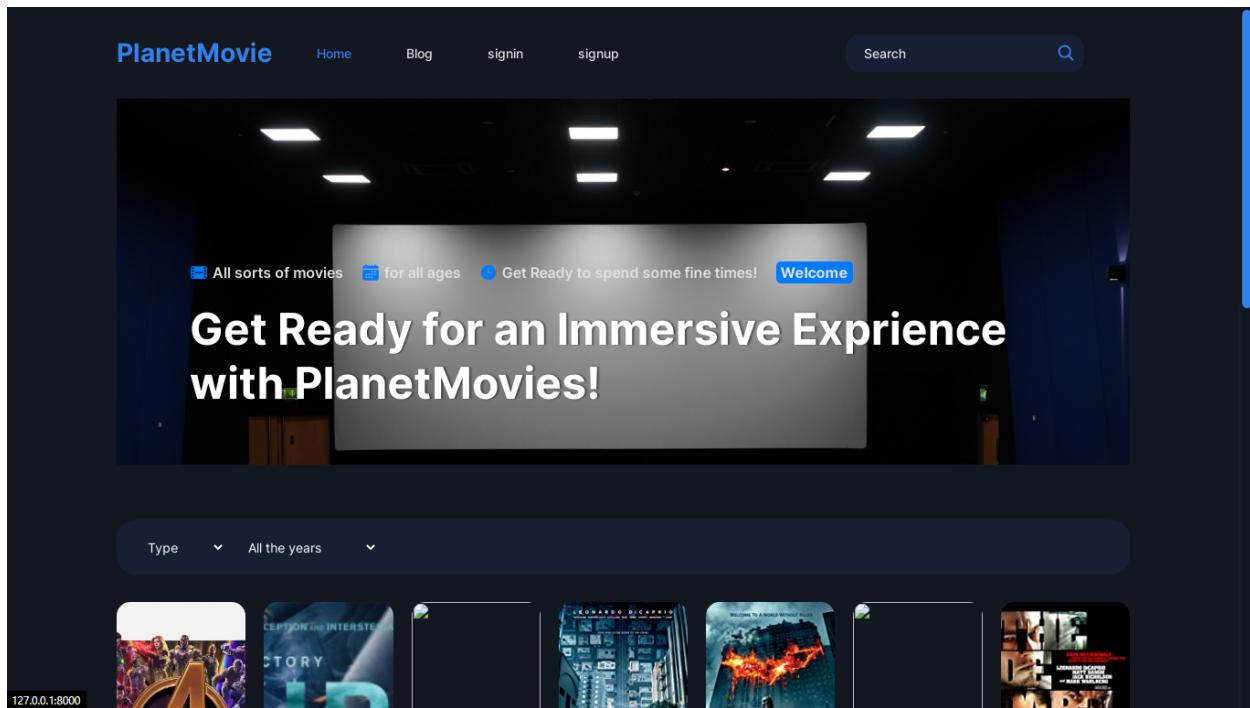
```
    added_at timestamp  
);
```

5. Pour la table "FavFilm":

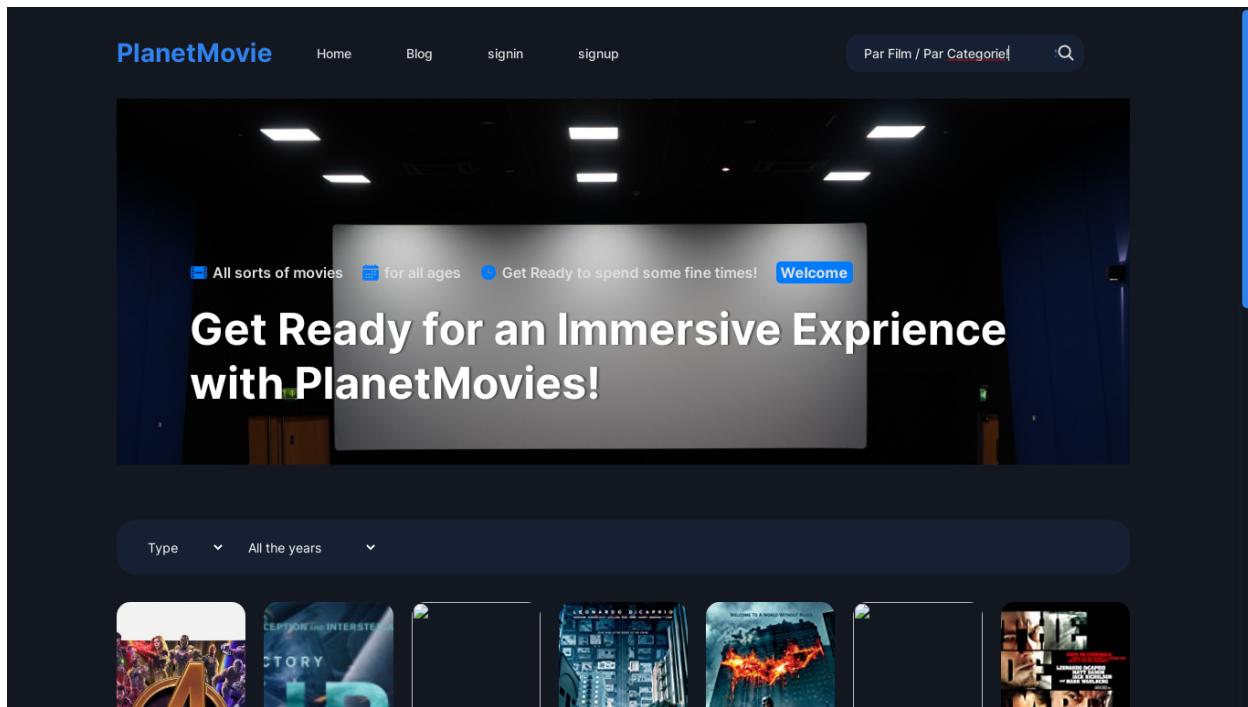
```
CREATE TABLE favfilm (  
    id uuid PRIMARY KEY,  
    user_id text,  
    film_id uuid,  
    added_at timestamp  
);
```

## Les différents modules de notre site web :

- La page principale :

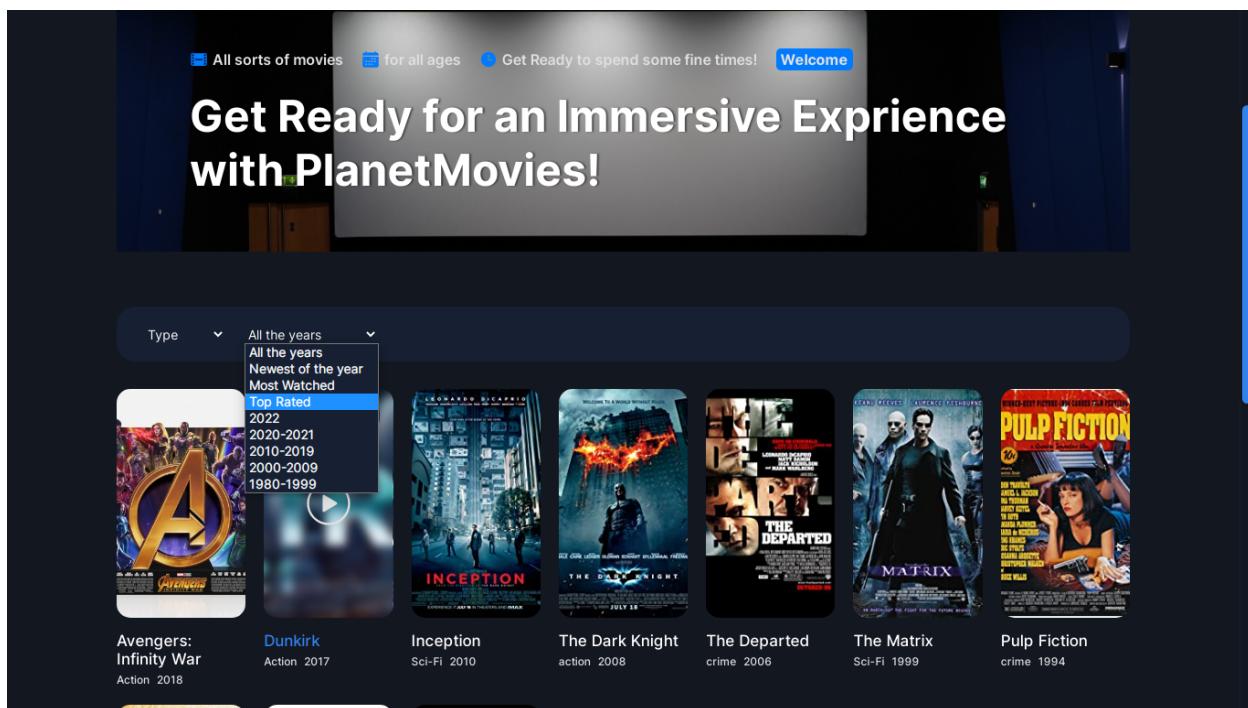


- La barre de recherche :

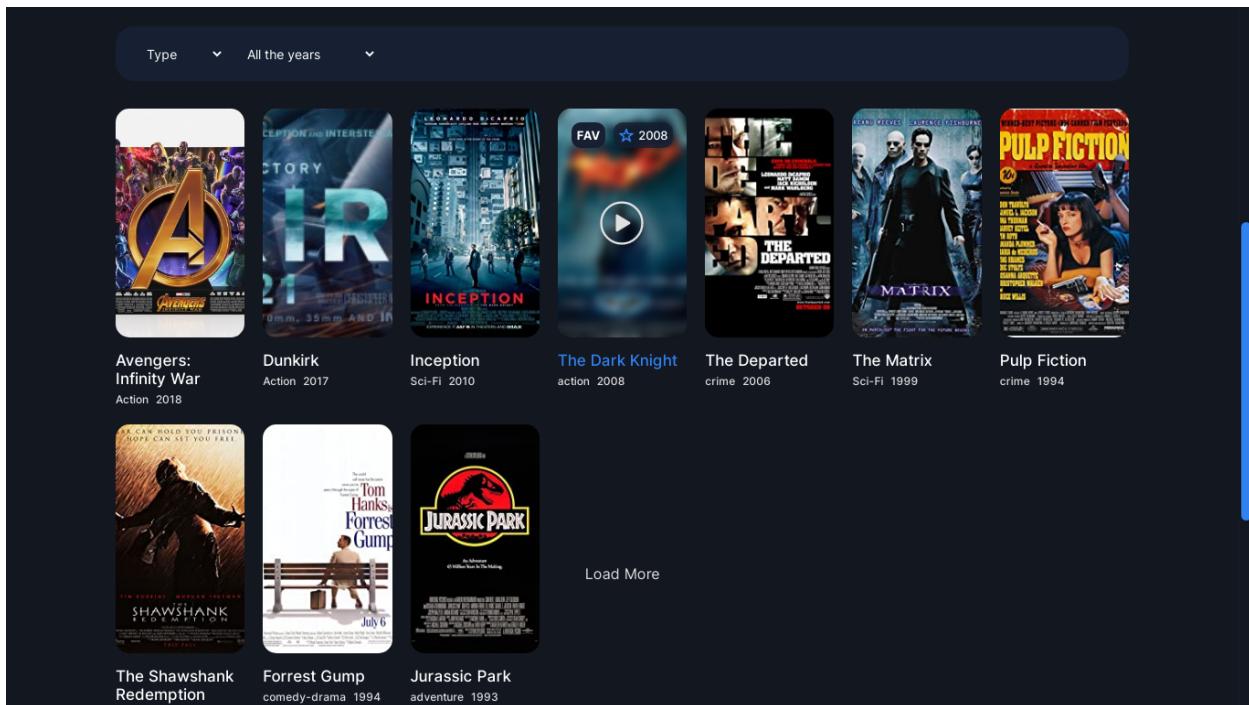


- La barre de recherche / Filtrage :

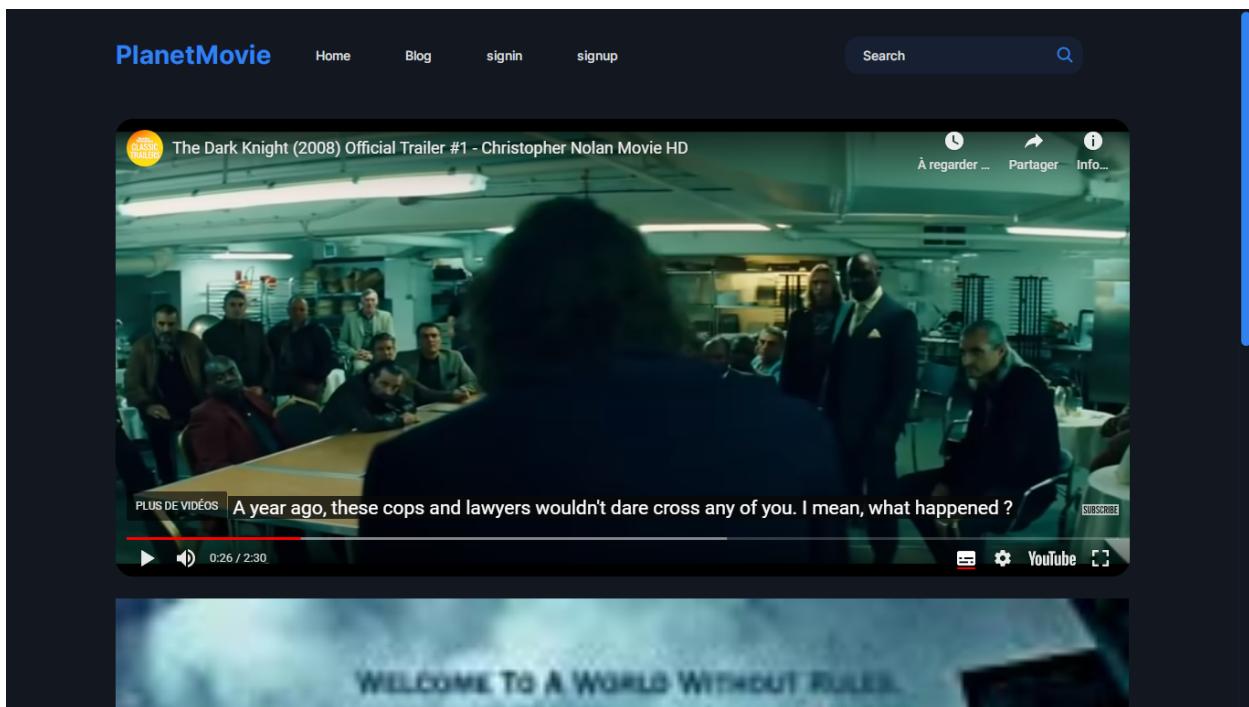
Exemple : Recherche Selons L'année de sortie des films :



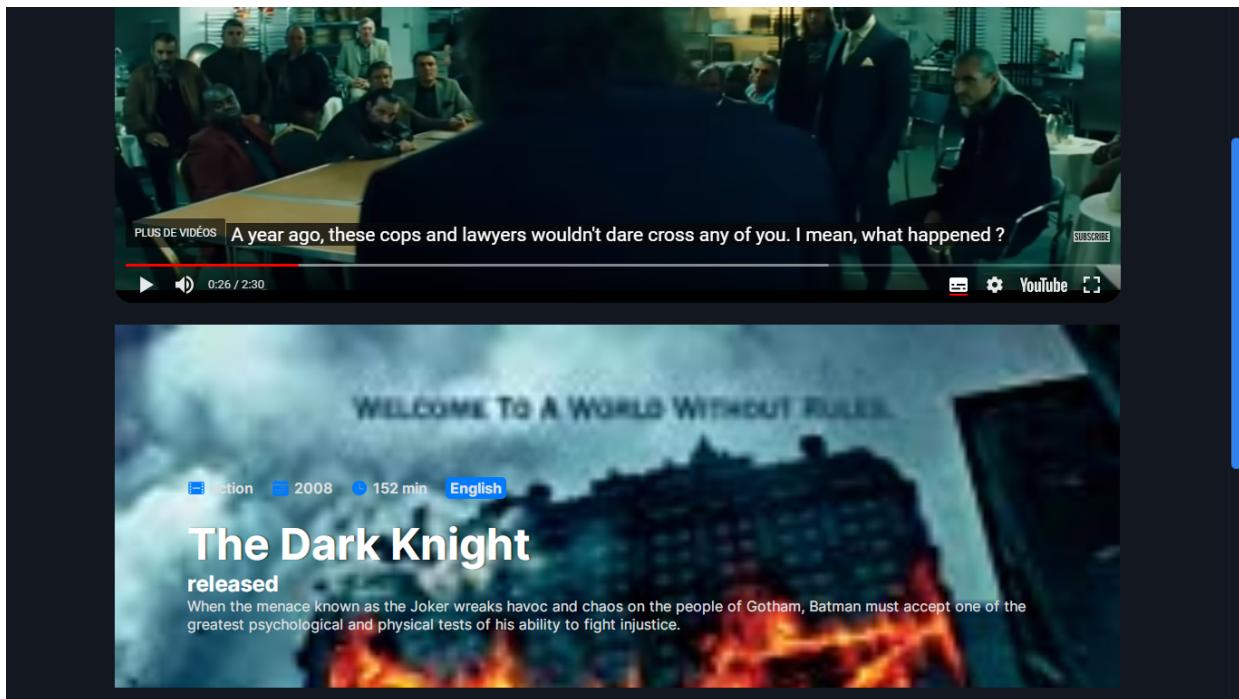
- La sections des films : Preview



- Affichage des informations détaillées de films :
  1. Trailer de film :



2. Les informations de films :



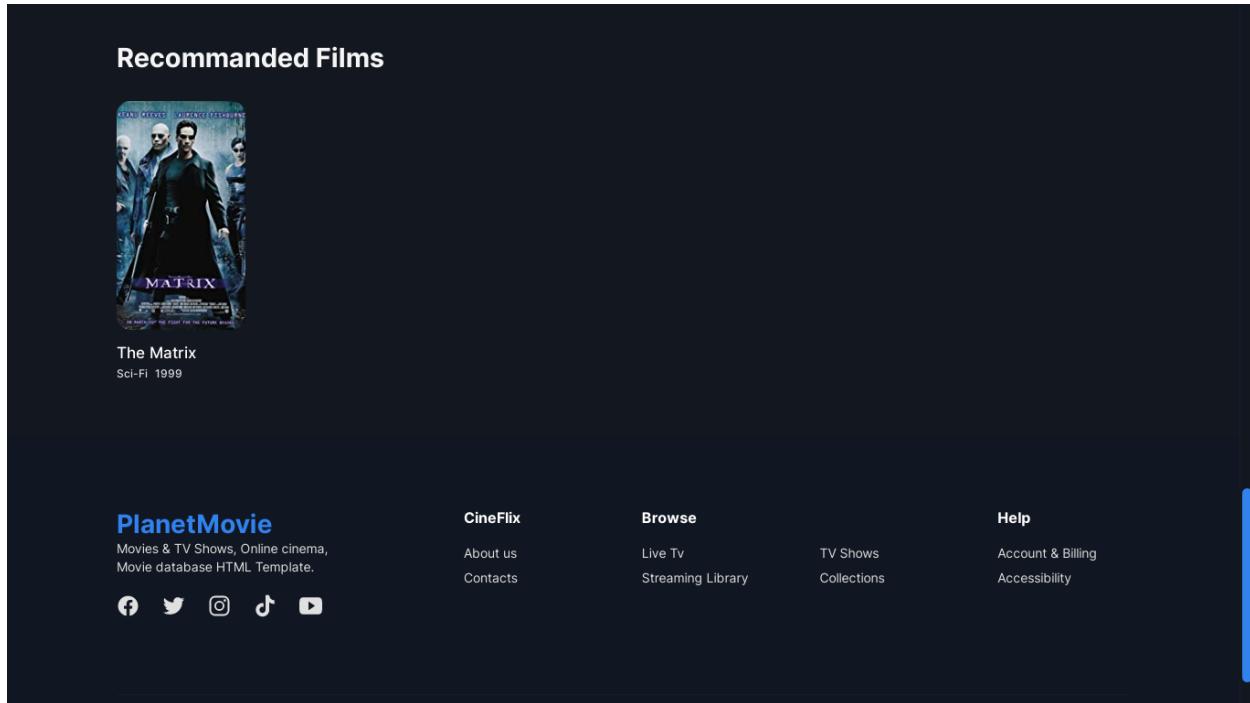
3. Section ou l'utilisations peut ajouter les commentaires/Section Commentaires

A screenshot of a website's comment section for the movie 'The Dark Knight'. The page has a dark theme. At the top, there is a banner with the movie's title and a subtitle: 'greatest psychological and physical tests of his ability to fight injustice.'. Below the banner, a heading says 'Leave a Comment'. There is a large input field with the placeholder 'Ajouter Un commentaire ! |' and a 'Submit' button below it. Underneath the input field, there is a heading 'Comments' followed by three user comments, each enclosed in a box:

- Utilisateur : aa**  
date :January 20, 2023  
Description de l'article : I like The plot Twist ! The principle actor did an amazing job in bringing the caractere to life , loved it !
- Utilisateur : aa**  
date :January 20, 2023  
Description de l'article : I agree with you , I recommend to you Avengers End game , i think you'll like @User2023!
- Utilisateur : User2023**  
date :January 20, 2023  
Description de l'article : I like too , great acting accross the board , wished it was longer!

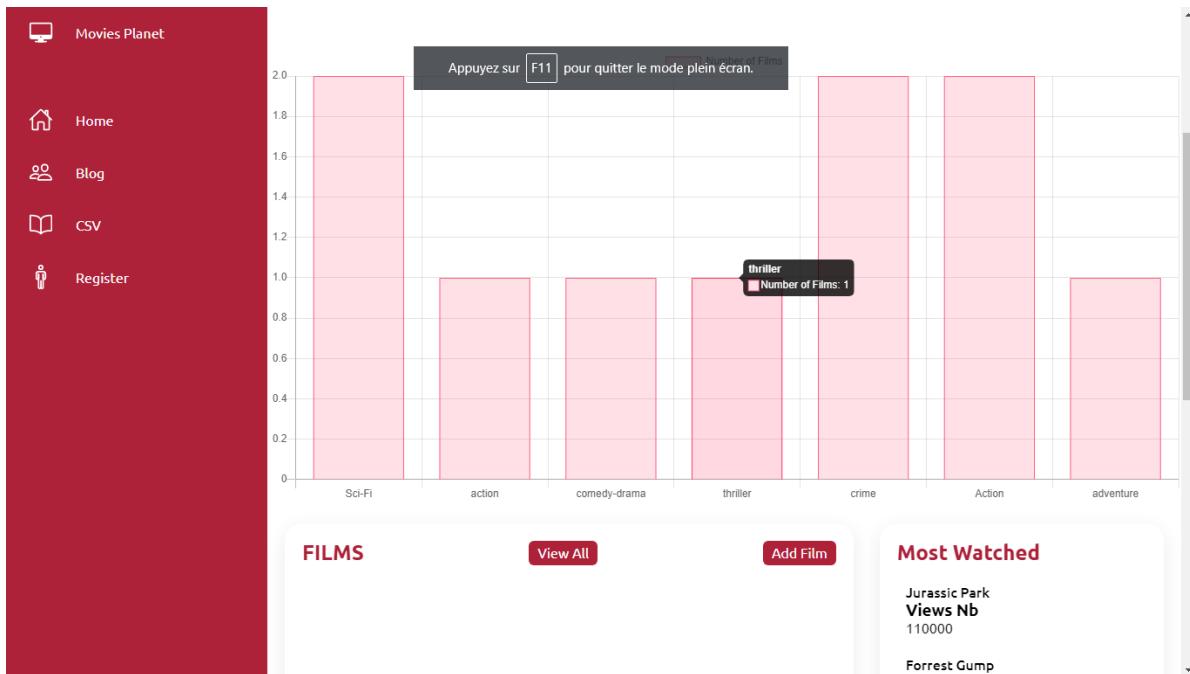
4. Section recommendations : ici a chaque fois qu'un utilisateur accède à un

certain film des recommandations sont faites sur la base du film qu'il a consulté.



## 5. Admin panel :

statistiques sur la base de donnée (nombres de films par catégorie) :



6. Ici l'admin a la possibilité d'imprimer sa base de données sous forme csv , il peut aussi ajouter , editer ou supprimer un film de la base de donnée :

The screenshot shows the 'Movies Planet' dashboard. The sidebar on the left includes 'CSV' in the list. The main area displays a table of films with columns: Number, Name, genre, year, status, and Modify/delete. The table lists seven films: Jurassic Park, Forrest Gump, The Shawshank Redemption, Pulp Fiction, The Matrix, The Departed, and The Dark Knight. To the right is a 'Most Watched' section with a list of movies and their view counts: Jurassic Park (110000), Forrest Gump (120000), The Shawshank Redemption (100000), Pulp Fiction (90000), The Matrix (70000), The Departed (60000), The Dark Knight (200000), and Inception (80000). A URL '127.0.0.1:8000/movie4\_csv' is visible at the bottom left.

- L'utilisateur peut consulter le blog et les dernières nouvelles à propos des films



## **House Of The Dragon: Why Are Targaryens The Only Dragonriders?**

In House of the Dragon, the Targaryens claim

- L'Admin peut Ajouter , supprimer , modifier les blogs :

Exemple d'ajout d'une article :

## **Create New Blog**

Create a new post below

title

|

body

# CONCLUSION

En conclusion, notre projet visait à étudier le modèle de données NoSQL orienté colonne et à mettre en place un livrable basé sur ce modèle. Au cours de ce projet, nous avons exploré les concepts clés du modèle, tels que les colonnes de partition et de clustering, et les avons mis en pratique en utilisant le framework Django avec l'extension Django Cassandra Engine. Nous avons également comparé les modèles de données SQL et NoSQL, soulignant les avantages et les inconvénients de chacun. Nous avons constaté que le modèle orienté colonne est particulièrement adapté pour les gros volumes de données et les requêtes hautement parallèles, tels que celles utilisées dans les applications de streaming de données en temps réel. En mettant en place notre livrable, nous avons rencontré des défis tels que l'intégration de différentes fonctionnalités telles que les commentaires des utilisateurs et la recommandation de films similaires, mais nous avons réussi à les surmonter grâce à la collaboration efficace de notre groupe. Notre projet nous a permis de comprendre les avantages et les limites du modèle orienté colonne et de mettre en pratique nos connaissances en utilisant un framework populaire comme Django. Nous espérons que les enseignements tirés de ce projet seront utiles pour les projets futurs et nous tenons à remercier notre professeur chargé de ce projet monsieur **Mohamed Essaid KHANOUCHÉ** pour son accompagnement et ses précieux conseils tout au long de ce travail et toutes les personnes qui ont contribué à la réalisation de ce projet.

1.